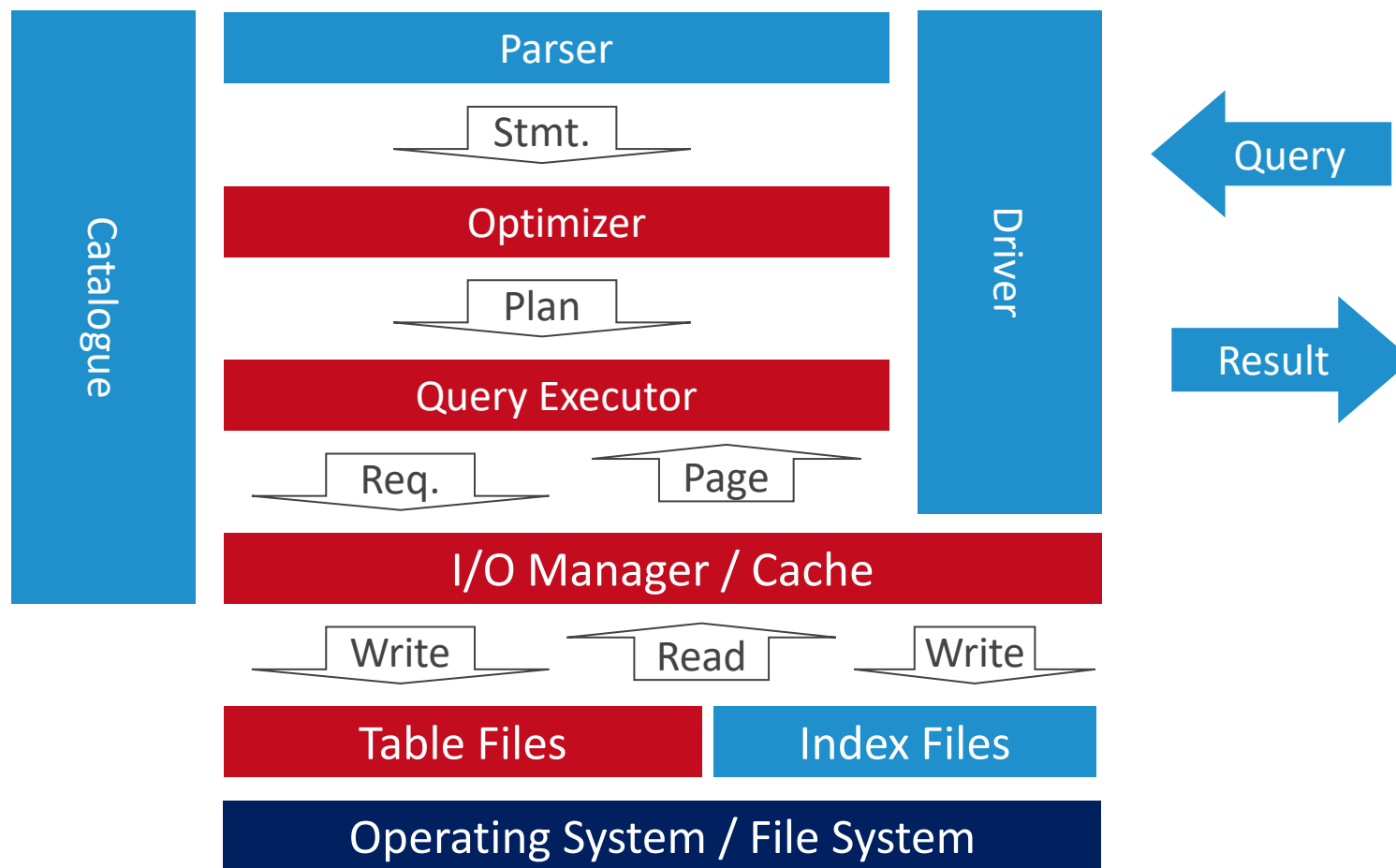


DBTLAB – Exercise 7: Query Optimization I – Join Order

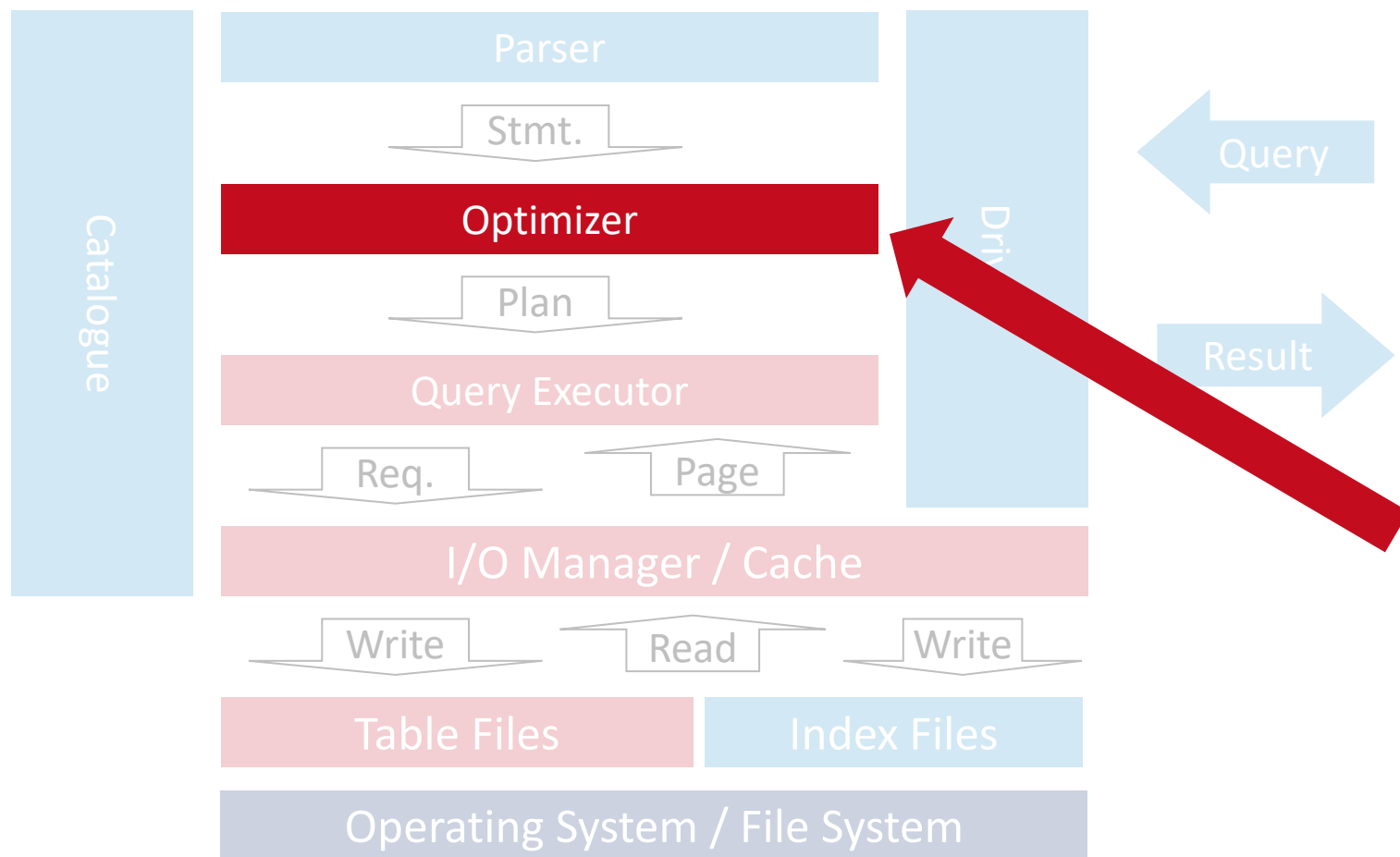
Rudi Poepel Lemaitre

based on material from Volker Markl, Sebastian Breß, Martin Kiefer, Viktor Rosenfeld

Basic system architecture



Where are we today?



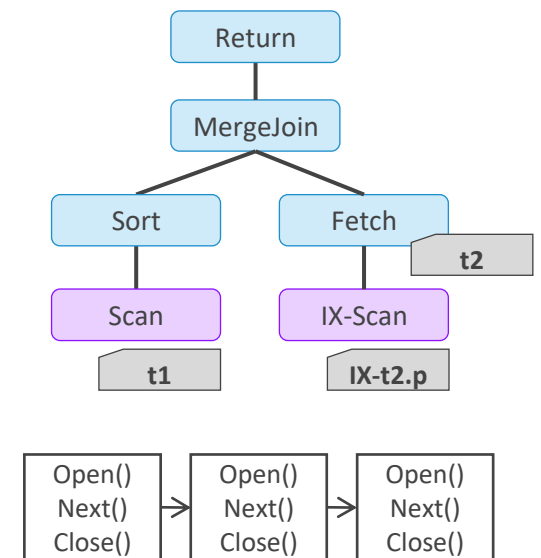
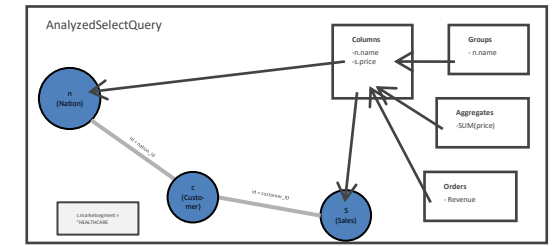
Select an “optimal” physical plan (avoid bad cases)
→ **“Reasonably-Good-imizer”**

Today: Join order optimization

Query Optimization

Creation of the Optimal Query Plan

- Input:
 - Semantically Analyzed Query
 - Statistics
- Output:
 - Query Execution Plan: Tree of Plan Operators
 - Plan operators describe relational properties, algorithms, parameters, and costs
- Next Step:
 - Convert the Query Execution Plan into an executable form
 - Compile it to interpreted code
 - Or create a runnable Iterator-Tree



Optimizer's Parameters

- Two essential parameters used to determine the optimal plan
 - Cardinalities
 - Operator Costs (estimated with the help of cardinalities)
- What cardinalities do we need to estimate?
 - Base cardinality for the tables
 - Selectivity of predicates
 - Selectivity of joins
 - Size of a grouped result

Optimizer's Parameters

- Cardinalities are estimated using statistics and a variety of methods
 - Stored in the system's catalogue
 - Obtained through statistics collection utilities
 - Obtained through query feedback
- What to do in the absence of statistics?
 - Use default values / fall back to heuristics
 - Sampling

Cost Estimation

- Costs of an Operator
 - I/O costs
 - CPU costs
 - Memory consumption
 - Network cost (for distributed database systems)
- Which costs are dominant for our system?

Design of our Optimizer

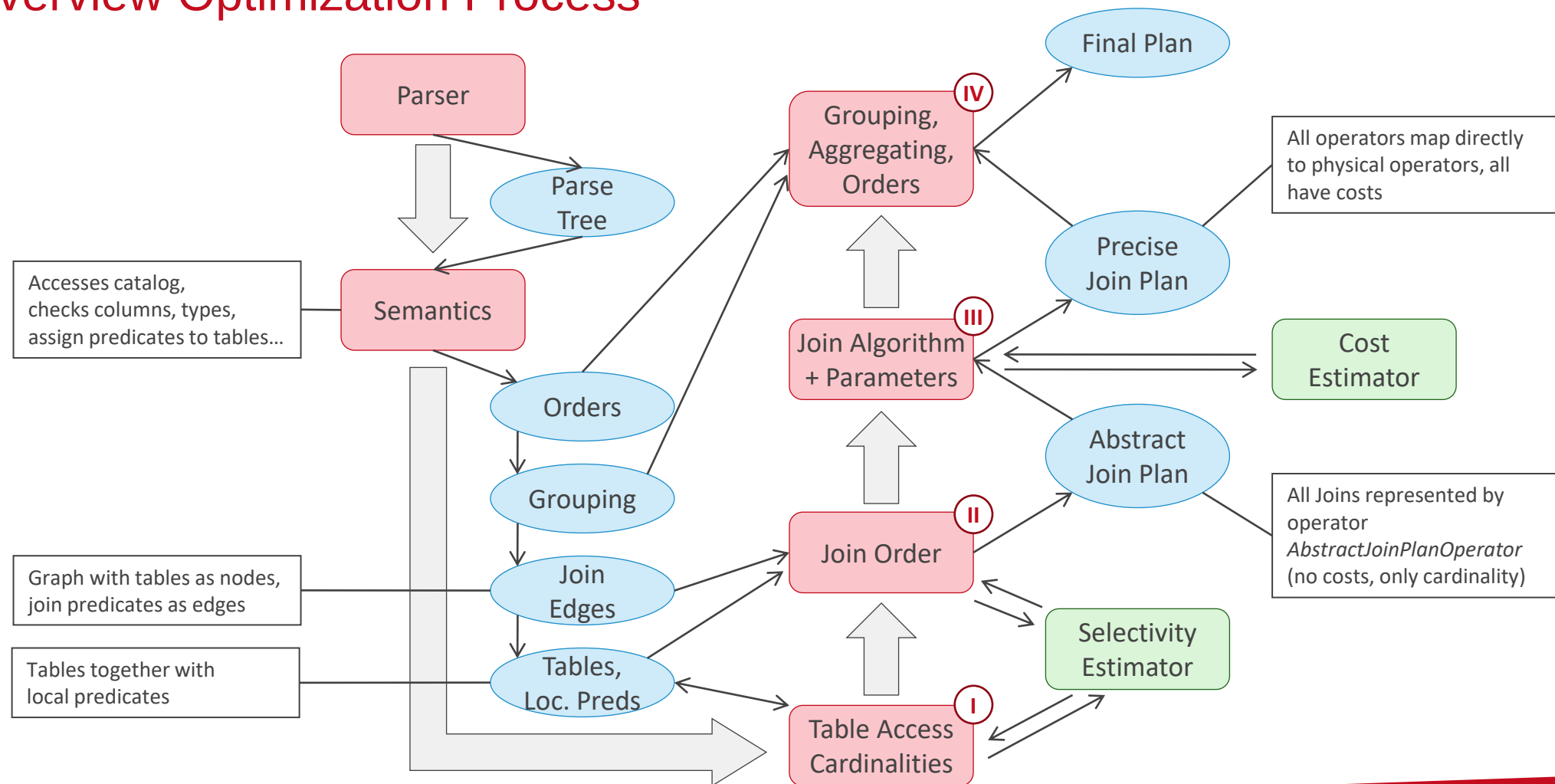
4 Steps Approach

- I) Estimation of local cardinalities
 - Cardinalities after single table access and its local predicates
- II) Cardinality-based selection of Join Order
 - Not optimal concerning total costs, but typically quite robust
 - Dynamic Programming: Optimal concerning cost formula (minimizing intermediate results)
- III) Selection of Table Access and Selection of Physical Join Operator
 - Based on I/O cost estimations
- IV) Generation of GroupBy and OrderBy
 - Just added on top of query, no analysis for push-down (sometimes aggregations are possible before the joins)

4 Steps Approach

- I) Estimation of local cardinalities
 - Cardinalities after single table access and its local predicates
- II) **Cardinality-based selection of Join Order**
 - Not optimal concerning total costs, but typically quite robust
 - Dynamic Programming: Optimal concerning cost formula (minimizing intermediate results)
- III) Selection of Table Access and Selection of Physical Join Operator
 - Based on I/O cost estimations
- IV) Generation of GroupBy and OrderBy
 - Just added on top of query, no analysis for push-down (sometimes aggregations are possible before the joins)

Overview Optimization Process

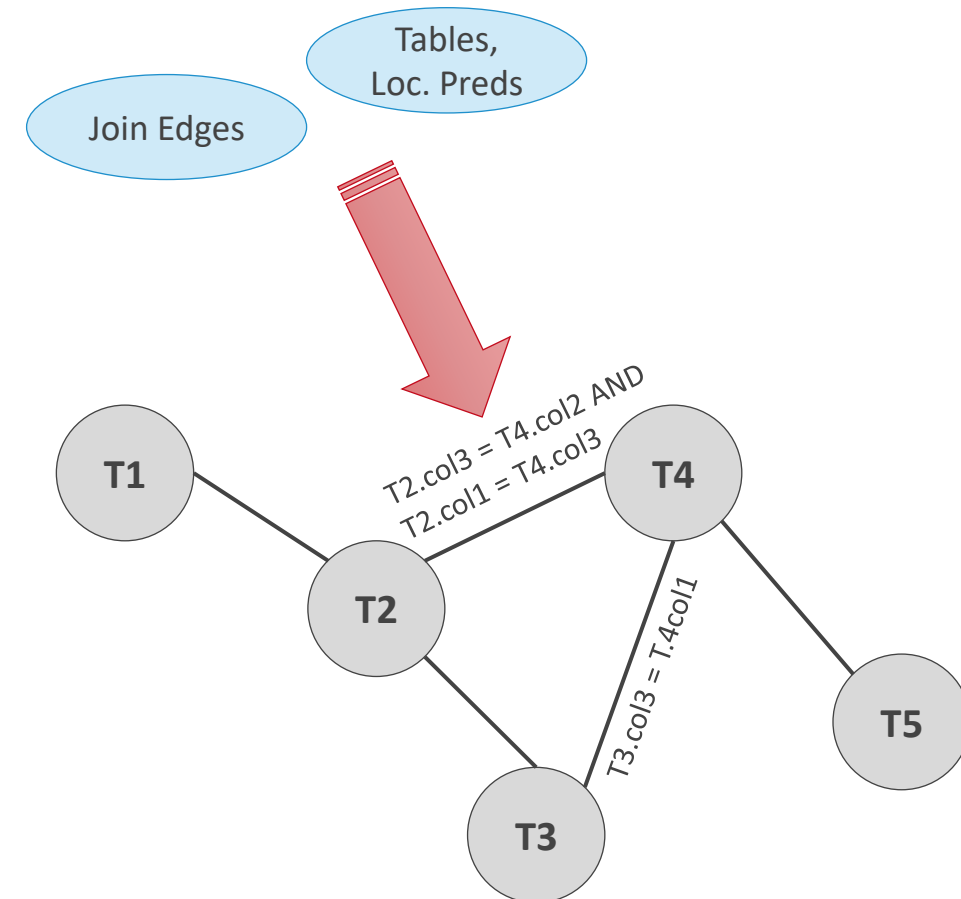


Phase I – Table Access Cardinalities

- Input: Relations
 - Contains reference to the base table (or subquery).
 - Contains the local predicate(s).
- 1) Estimate the selectivity of the predicate(s).
 - May be a boolean atom or a conjunction or disjunction or an arbitrary combination.
- 2) Estimate result cardinality.
- The *Relation* logical operator is used as a placeholder representing the relational properties after the relation access.
 - Later replaced by table-scan / index-scan / index-seek + fetch.

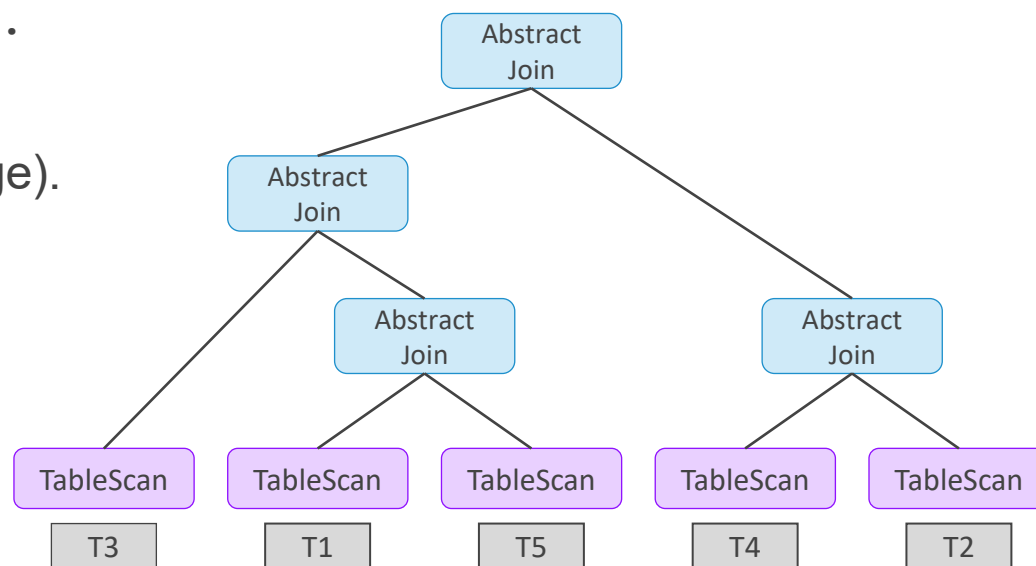
Phase II – Join Order

- Input: Graph representing the joins.
- Nodes
 - Represent Tables (Relations).
 - Have cardinalities from the previous phase.
- Edges
 - Reference the tables that they join.
 - Contain the join predicate(s).



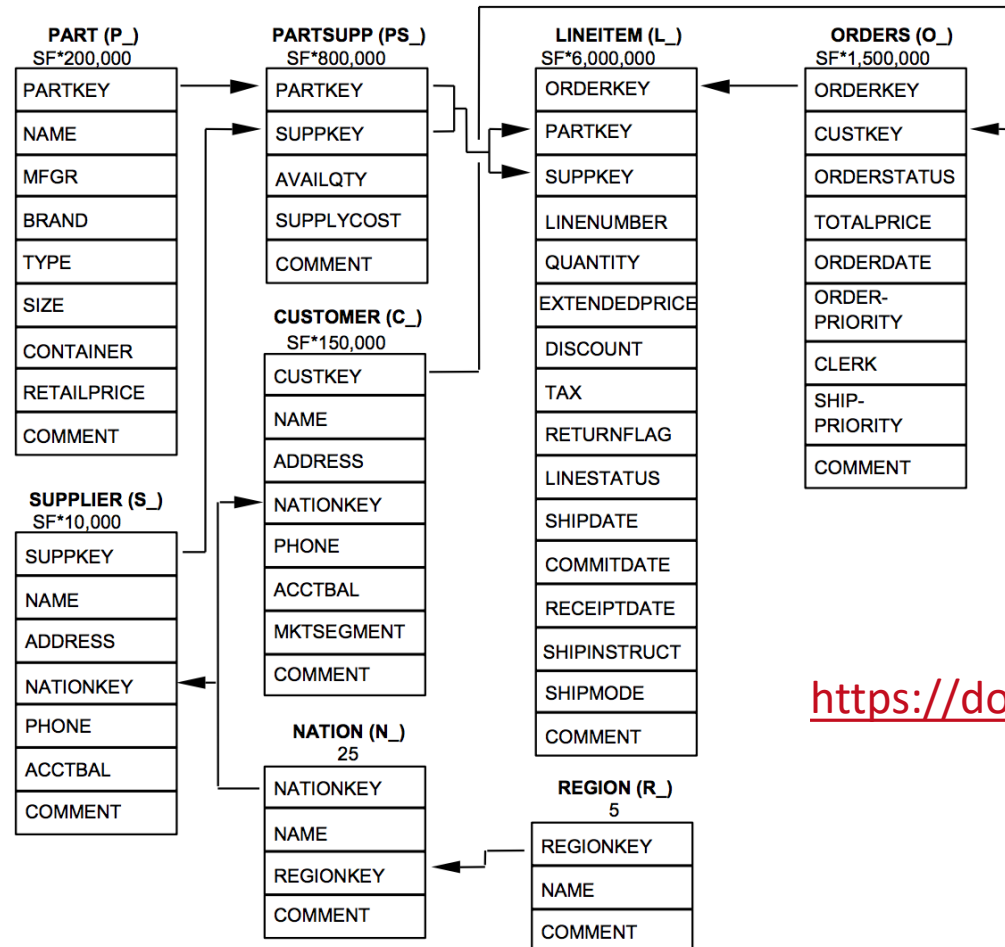
Phase II – Join Order

- Output: Query Plan, consisting of the given Relations (nodes from the graph) and **AbstractJoinPlanOperator**.
- In the course of enumerating plans, create every join as an abstract **AbstractJoinPlanOperator**.
- **AbstractJoinPlanOperator** contains an predicate (as was associated with the join's edge).
- Right/Left child assignment arbitrary at that point.
- After creation of a new operator, its costs are evaluated using **CardinalityEstimator** class.



Join Order Selection – Dynamic Programming Algorithm

Schema (TPC-H)

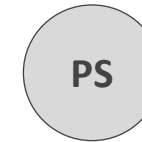
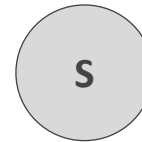


<https://docs.snowflake.com/en/user-guide/sample-data-tpch.html>

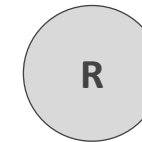
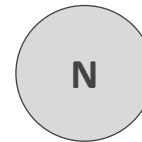
Query and Join Graph

```

SELECT ps.*, s.*, n.*, r.*
FROM   partsupp ps,
       supplier s,
       nation n,
       region r
WHERE  ps.suppkey = s.suppkey
      AND s.nationkey = n.nationkey
      AND n.regionkey = r.regionkey
      AND r.name = "AFRICA"
      AND ps.partkey > 1000
      AND ps.partkey < 1005;
  
```



Every requested table
is a node



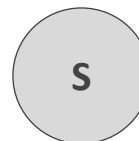
Query and Join Graph

```

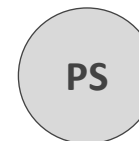
SELECT ps.*, s.*, n.*, r.*
FROM   partsupp ps,
       supplier s,
       nation n,
       region r
WHERE  ps.suppkey = s.suppkey
      AND s.nationkey = n.nationkey
      AND n.regionkey = r.regionkey
      AND r.name = "AFRICA"
      AND ps.partkey > 1000
      AND ps.partkey < 1005;

```

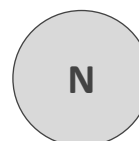
CARD = 200



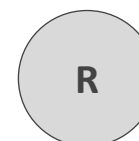
CARD = 20



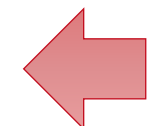
Node cardinalities
honor local predicate
selectivity



CARD = 25



CARD = 5
CARD = 1



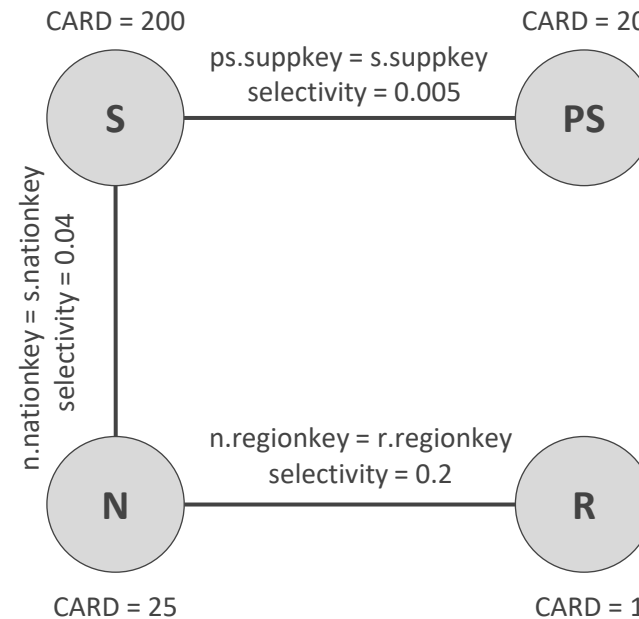
name
"AFRICA "
"AMERICA"
"ASIA"
"EUROPE"
"MIDDLE EAST"

Query and Join Graph

```

SELECT ps.*, s.*, n.*, r.*
FROM   partsupp ps,
       supplier s,
       nation n,
       region r
WHERE  ps.suppkey = s.suppkey
      AND s.nationkey = n.nationkey
      AND n.regionkey = r.regionkey
      AND r.name = "AFRICA"
      AND ps.partkey > 1000
      AND ps.partkey < 1005;

```



Edges symbolize the join predicates with their selectivity

Join Order Selection – Dynamic Programming Algorithm

- Step 1: Initialize the table with bitmaps and costs for a single relation

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0

- The first entries in the table are just the base relations.
- The cost of base table access is 0.

Join Order Selection – Dynamic Programming Algorithm

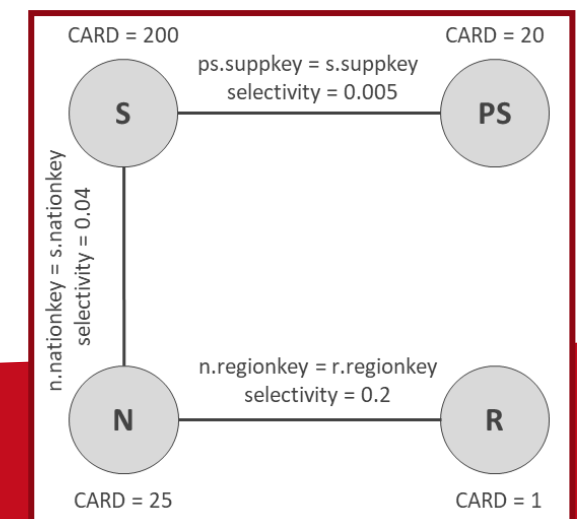
- Step 2: Add joins of single tables

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)



Join predicate	Relation / Join Bitmap
$PS.supkey = S.supkey$	0011 (3)
$N.nationkey = S.nationkey$	0101 (5)
$R.regionkey = N.regionkey$	1100 (12)

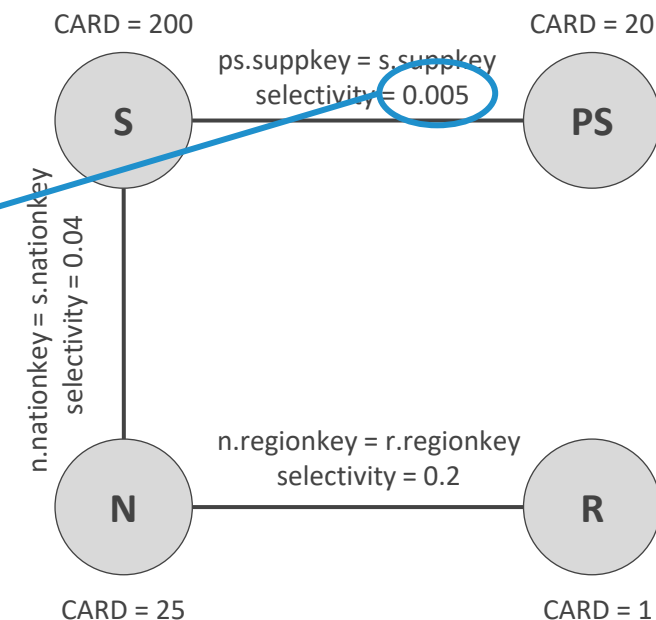
- There are 6 possible combinations of joining 4 tables.
- However, only three combinations match a join predicate of the query.
- Only these three are added to the table.



Join Order Selection – Dynamic Programming Algorithm

– Computing output cardinality

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20 (200 * 20 * 0.005)	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)



You **DON'T** need to implement this yourself... Use the provided **CardinalityEstimator** object!

Join Order Selection – Dynamic Programming Algorithm

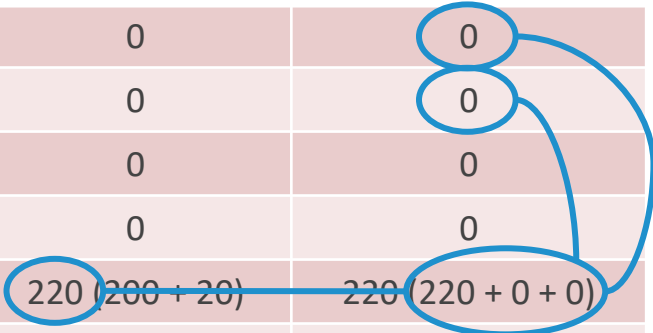
– Computing the operator costs

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)

Join Order Selection – Dynamic Programming Algorithm

– Computing cumulative costs

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)



Join Order Selection – Dynamic Programming Algorithm

- Step 4: Add joins of 2 tables + 1 table

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)

- Based on the join predicates, we can generate three plans joining three tables.
 - Two plans join S, N, and R (bitmap 1101).
 - Two plans join P, S, and N (bitmap 0111).

Join Order Selection – Dynamic Programming Algorithm

– Step 4: Add joins of 3 tables + 1 table

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)

– Based on the join predicates, we can generate three plans joining three tables.

- Two plans join S, N, and R (bitmap 1101).
- Two plans join P, S, and N (bitmap 0111).

– We purge the slower plan for bitmap 0111.

← Purged
445 > 265

Join Order Selection – Dynamic Programming Algorithm

- Step 4: Add joins of 3 tables + 1 table

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)

- Based on the join predicates, we can generate three plans joining three tables.
 - Two plans join S, N, and R (bitmap 1101).
 - Two plans join P, S, and N (bitmap 0111).
- We purge the slower plan for bitmap 1101.

← Purged
426 > 231

Join Order Selection – Dynamic Programming Algorithm

– Step 4: Add joins of 3 tables + 1 table

– Based on the join predicates, we can generate two plans joining 3 + 1 tables.

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)
$((N \bowtie R) \bowtie S) \bowtie PS$	1111 (15)	4	60 (40 + 20)	291 (60 + 231 + 0)
$((S \bowtie PS) \bowtie N) \bowtie R$	1111 (15)	4	21 (20 + 1)	286 (21 + 265 + 0)

Join Order Selection – Dynamic Programming Algorithm

- Step 4: Add joins of 3 tables + 1 table

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)
$((N \bowtie R) \bowtie S) \bowtie PS$	1111 (15)	4	60 (40 + 20)	291 (60 + 231 + 0)
$((S \bowtie PS) \bowtie N) \bowtie R$	1111 (15)	4	21 (20 + 1)	286 (21 + 265 + 0)

- Based on the join predicates, we can generate two plans joining 3 + 1 tables.
- We purge the slower plan.

← Purged
291 > 286

Join Order Selection – Dynamic Programming Algorithm

– Step 5: Add joins of 2 tables + 2 tables

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)
$((N \bowtie R) \bowtie S) \bowtie PS$	1111 (15)	4	60 (40 + 20)	291 (60 + 231 + 0)
$((S \bowtie PS) \bowtie N) \bowtie R$	1111 (15)	4	21 (20 + 1)	286 (21 + 265 + 0)
$(S \bowtie PS) \bowtie (N \bowtie R)$	1111 (15)	4	25 (20 + 5)	271 (25 + 220 + 26)

– So far, we have considered left-deep plans.

– Based on the join predicates, we can also generate a bushy plan that joins all four tables.

Join Order Selection – Dynamic Programming Algorithm

– Step 5: Add joins of 2 tables + 2 tables

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)
$((N \bowtie R) \bowtie S) \bowtie PS$	1111 (15)	4	60 (40 + 20)	291 (60 + 231 + 0)
$((S \bowtie PS) \bowtie N) \bowtie R$	1111 (15)	4	21 (20 + 1)	286 (21 + 265 + 0)
$(S \bowtie PS) \bowtie (N \bowtie R)$	1111 (15)	4	25 (20 + 5)	271 (25 + 220 + 26)

– So far, we have considered left-deep plans.

– Based on the join predicates, we can also generate a bushy plan that joins all four tables.

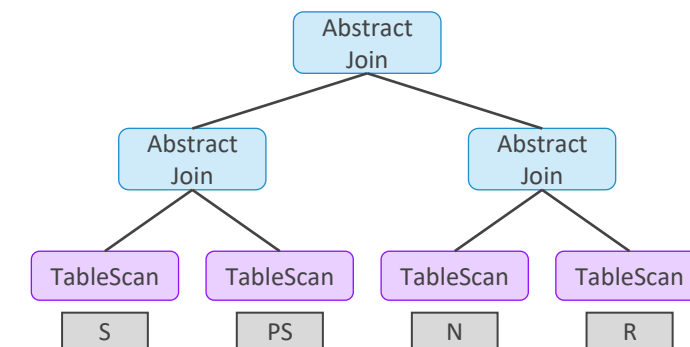
Purged
286 > 271

Join Order Selection – Dynamic Programming Algorithm

- Step 5: Add joins of 2 tables + 2 tables

Plan	Relation / Join Bitmap	Output Cardinality	Operator Costs	Cumulative Costs
S	0001 (1)	200	0	0
PS	0010 (2)	20	0	0
N	0100 (4)	25	0	0
R	1000 (8)	1	0	0
$S \bowtie PS$	0011 (3)	20	220 (200 + 20)	220 (220 + 0 + 0)
$S \bowtie N$	0101 (5)	200	225 (200 + 25)	225 (225 + 0 + 0)
$N \bowtie R$	1100 (12)	5	26 (25 + 1)	26 (26 + 0 + 0)
$(N \bowtie R) \bowtie S$	1101 (13)	40	205 (200 + 5)	231 (205 + 26 + 0)
$(S \bowtie N) \bowtie PS$	0111 (7)	20	220 (200 + 20)	445 (220 + 225 + 0)
$(S \bowtie PS) \bowtie N$	0111 (7)	20	45 (20 + 25)	265 (45 + 220 + 0)
$(S \bowtie N) \bowtie R$	1101 (13)	40	201 (200 + 1)	426 (201 + 225 + 0)
$((N \bowtie R) \bowtie S) \bowtie PS$	1111 (15)	4	60 (40 + 20)	291 (60 + 231 + 0)
$((S \bowtie PS) \bowtie N) \bowtie R$	1111 (15)	4	21 (20 + 1)	286 (21 + 265 + 0)
$(S \bowtie PS) \bowtie (N \bowtie R)$	1111 (15)	4	25 (20 + 5)	271 (25 + 220 + 26)

- So far, we have considered left-deep plans.
- Based on the join predicates, we can also generate a bushy plan that joins all four tables.
- This is the fastest plan.



Best plan (lowest cumulative cost)

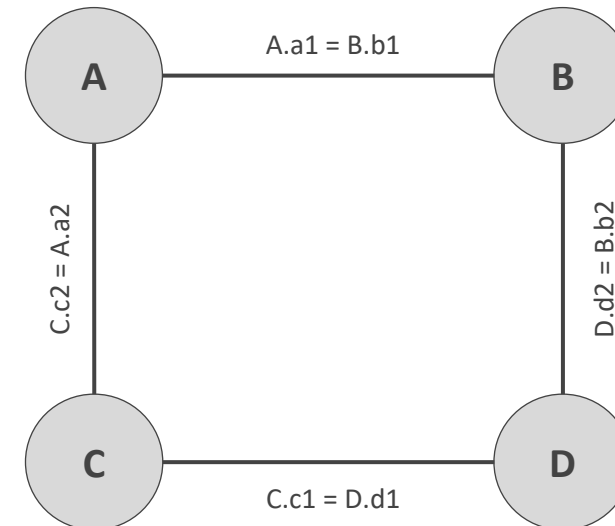
Join Order Optimization – General Description and Hints

Exercise 7 - General Information

- Implement the following functionality
 - Implement the Join-Order Optimizer
 - Must implement the Dynamic Programming Algorithm as described in the lecture and the book
 - Join order estimator must estimate intermediate costs (= cardinalities)
 - The cardinality estimator is given
- The **Operator Cost** of a base table access is zero because we do not know if it will be a table scan or index scan.
- The **Operator Costs** of a join are defined as the sum of the output cardinalities of the join inputs
- The **Cumulative Costs** of a sub-plan (join or base table access) are defined as the sum of the **Operator Costs** of the root and the **Cumulative Costs** of the children of the plan

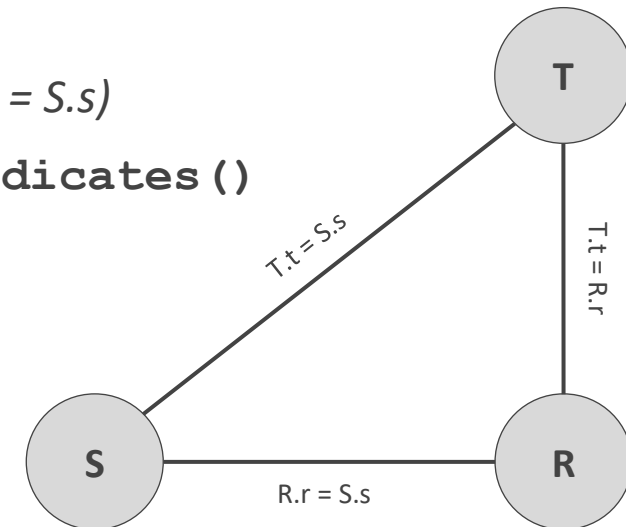
Join Predicates Hint

- Note that you might have to construct the join predicates as a conjunct of all join edges
- Consider the join graph below
 - If you want to construct $(A \bowtie_q B) \bowtie_p C$, the join predicate p is the AC edge, i.e. $p := (C.c2 = A.a2)$
 - However, if you want to construct $(A \bowtie_{q1} B) \bowtie_p (C \bowtie_{q2} D)$, the join predicate p is the conjunct of all cutting edges, i.e.
 $p := (C.c2 = A.a2 \text{ AND } D.d2 = B.b2)$



Join Predicates Hint

- The conjunctions may contain junk atoms
 - In these cases, you should prune the junk as follows
- Consider the join graph below
 - If you want to construct $(R \bowtie_q S) \bowtie_p T$, the join predicate p is given by the TR and TS edges, i.e. $p := (T.t = R.r \text{ AND } T.t = S.s)$
 - However, $R.r = S.s$ is already ensured by the $(R \bowtie_q S)$ subplan since $q := (R.r = S.s)$, so it suffices to specify either $p := (T.t = R.r)$ or $p := (T.t = S.s)$
 - **HINT:** Use the `JoinOrderOptimizerUtils#filterTwinPredicates()` method to filter the junk atoms from the constructed compound join predicates in a consistent way



Cardinality Estimator

- The constructor for the join order optimizer receives a `CardinalityEstimator`.
- Use `CardinalityEstimator#estimateJoinCardinality` to estimate the cardinalities from the new abstract joins you construct.
- To get the cardinalities from any operator, use the `getOutputCardinality` method that is available for all operators extending the `OptimizerPlanOperator` abstract class.

Exercise 7

Exercise 7

- Implement a join order optimizer using the following interface:
 - `de.tuberlin.dima.minidb.optimizer.joins.JoinOrderOptimizer`
- Implement the following factory method:
 - `createJoinOrderOptimizer`

Tests and points

Test name	Points	Description
testOptimizerJoinOf4WithPreds	2	<i>Tests simple join of Nation, Region, ParSupp, and Supplier. Predicate on Region.</i>
testOptimizerJoinOf4	1	<i>Tests simple join of ParSupp, Supplier, Lineitem and Orders. No predicates.</i>
testOptimizerJoinOf8	3	<i>Tests simple join of Region, Nation, Customer, Orders, Lineitem, ParSupp, Supplier, and Part. No predicates.</i>
testOptimizerJoinWithSubquery	3	Tests a complex join query with a subquery (same as the student's test).
testOptimizerJoinOf4WithPreds2	2	<i>Tests simple join of Nation, Region, ParSupp, and Supplier. Range predicate on ParSupp.</i>
testOptimizerJoinWithSubquery2	3	Tests a complex join query with a subquery (different from the student's test).